СИСТЕМЫ ТЕХНИЧЕСКОГО ЗРЕНИЯ

БИБЛИОТЕКА **EIGEN**. ИТЕРАЦИОННЫЕ МЕТОДЫ. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ.

EIGEN – ЧУТЬ ПОДРОБНЕЕ

- На предыдущем занятии мы выяснили, как
  - Создавать вектора и матрицы в библиотеке Eigen
  - С помощью библиотеки найти решение, используя LU-разложение
- Теперь попробуем посмотреть чуть подробнее

EIGEN – РАЗЛИЧНЫЕ МЕТОДЫ РЕШЕНИЯ

- FullPivLU – LU-разложение с полным выбором главного элемента
- PartialPivLU - LU-разложение с выбором главного элемента по строкам
- HouseholderQR – QR-разложение
- LLT – метод Холецкого
- …

А.А. Амосов, Ю.А. Дубинский, Н.В. Копчёнова «Вычислительные методы для инженеров. Учебное пособие» - М., Высшая школа, 1994г.
М.П. Галанин, Е.Б. Савенков «Методы численного анализа математических моделей» - М., Изд-во МГТУ им. Н.Э. Баумана, 2018 http://ebooks.bmstu.ru/catalog/95/book1824.html

```cpp
#include <iostream>
#include <eigen3/Eigen/LU>
using namespace std;

int main(int argc, char *argv[])
{
   Eigen::Matrix4d A, lower, upper;
   Eigen::Vector4d b, x;

   A << 10, 6, 2, 0, 5, 1, -2, 4, 3, 5, 1, -1, 0, 6, -2, 2;
   b << 25, 14, 10, 8;

   Eigen::PartialPivLU<Eigen::Matrix4d> lu(A);

   cout << "Matrix A" << endl << A << endl;
   cout << "Vector b" << endl << b << endl;
   cout << "LU-decomp" << endl << lu.matrixLU() << endl;

   lower = lu.matrixLU().triangularView<Eigen::StrictlyLower>();
   lower = lower + Eigen::Matrix4d::Identity();
   upper = lu.matrixLU().triangularView<Eigen::Upper>();

   cout << "Matrix lower" << endl << lower << endl;
   cout << "Matrix upper" << endl << upper << endl;

   cout << "Multiplication" << endl << lower*upper << endl;
   cout << "Multiplication with permutation" << endl
      << lu.permutationP().inverse()*lower*upper << endl;

   return 0;
}
```

EIGEN – LU-РАЗЛОЖЕНИЕ

matrixLU возвращает квадратную матрицу состоящую из L и U

triangularView<>() возвращает матрицу в виде либо содержащим диагональ:

<Lower> <Upper>

Либо без неё:

<StrictlyLower> <StrictlyUpper>

$$A = P^{-1}LU,$$

где P^{-1} - матрица перестановки строк

```
Matrix A
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
Vector b
25
14
10
 8
LU-decomp
       10         6         2         0
        0         6        -2         2
      0.5 -0.333333  -3.66667   4.66667
      0.3  0.533333      -0.4      -0.2
Matrix lower
        1         0         0         0
        0         1         0         0
      0.5 -0.333333         1         0
      0.3  0.533333      -0.4         1
Matrix upper
      10        6        2        0
       0        6       -2        2
       0        0 -3.66667  4.66667
       0        0        0     -0.2
Multiplication
10  6  2  0
 0  6 -2  2
 5  1 -2  4
 3  5  1 -1
Multiplication with permutation
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
```

EIGEN – LU-РАЗЛОЖЕНИЕ

```cpp
#include <iostream>
#include <Eigen/LU>
using namespace std;

int main(int argc, char *argv[])
{
    Eigen::Matrix4d A, lower, upper;
    Eigen::Vector4d b, x;
    A << 10, 6, 2, 0, 5, 1, -2, 4, 3, 5, 1, -1, 0, 6, -2, 2;
    b << 25, 14, 10, 8;
    Eigen::FullPivLU<Eigen::Matrix4d> lu(A);
    cout << "Matrix A" << endl << A << endl;
    cout << "Vector b" << endl << b << endl;
    cout << "LU-decomp" << endl << lu.matrixLU() << endl;
    lower = lu.matrixLU().triangularView<Eigen::StrictlyLower>();
    lower = lower + Eigen::Matrix4d::Identity();
    upper = lu.matrixLU().triangularView<Eigen::Upper>();
    cout << "Matrix lower" << endl << lower << endl;
    cout << "Matrix upper" << endl << upper << endl;
    cout << "Multiplication" << endl << lower*upper << endl;
    cout << "Multiplication with permutation" << endl
         << lu.permutationP().inverse()*lower*upper*lu.permutationQ().inverse();
    return 0;
}
```

ПОЛНЫЙ ВЫБОР ГЛАВНОГО ЭЛЕМЕНТА

$$A = P^{-1}LUQ^{-1}$$

Здесь Q^{-1} - перестановка столбцов

```
Matrix A
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
Vector b
25
14
10
 8
LU-decomp
       10         6         0         2
        0         6         2        -2
      0.5 -0.333333   4.66667  -3.66667
      0.3  0.533333 -0.442857 -0.157143
Matrix lower
        1         0         0         0
        0         1         0         0
      0.5 -0.333333         1         0
      0.3  0.533333 -0.442857         1
Matrix upper
       10         6         0         2
        0         6         2        -2
        0         0   4.66667  -3.66667
        0         0         0 -0.157143
Multiplication
10  6  0  2
 0  6  2 -2
 5  1  4 -2
 3  5 -1  1
Multiplication with permutation
10  6  2  0
 5  1 -2  4
 3  5  1 -1
 0  6 -2  2
```

МЕТОД ХОЛЕЦКОГО (КВАДРАТНОГО КОРНЯ)

- Частный случай – матрица A является симметричной ($A^T = A$) и положительно определённой ($(Ax, x) > 0$)
- Таких случаев, на самом деле, довольно много
- Можно искать специальное представление

$$A = L \cdot L^T$$

- Метод примерно вдвое быстрее метода Гаусса, и гарантировано точнее за счёт меньшего накопления ошибки

```cpp
#include <iostream>
#define _USE_PREDEFINED_CONSTANT
#include <cmath>

#include <eigen3/Eigen/LU>
#include <eigen3/Eigen/QR>
#include <eigen3/Eigen/SVD>
#include <eigen3/Eigen/Dense>
#include <eigen3/Eigen/Core>
using std::cout;
using std::endl;
using Eigen::MatrixXf;
using Eigen::VectorXf;

#define SYS_SIZE 3

int main()
{
    MatrixXf  a(SYS_SIZE,SYS_SIZE);

    a << 8.1041, 0.0083, -3.1433, 1.7114, -0.0200, -0.6638, 4.2800, -0.0078, -1.6600;
//    a = Eigen::MatrixXf::Random(SYS_SIZE, SYS_SIZE);
    cout << "Matrix A\n" << a << endl;
    VectorXf b(SYS_SIZE), xplu(SYS_SIZE),  xflu(SYS_SIZE), xqr(SYS_SIZE);
    b << -4, 6.00, -8.5;
    //        b = Eigen::VectorXf::Random(SYS_SIZE);;
    Eigen::PartialPivLU<MatrixXf> plu(a);
    xplu= plu.solve(b);
    cout << "Partial pivoting solution\n" << xplu << endl;
    Eigen::FullPivLU<MatrixXf> flu(a);
    xflu= flu.solve(b);
    cout << "Full pivoting solution\n" << xflu << endl;
    Eigen::FullPivHouseholderQR<MatrixXf> qr(a);
    xqr= qr.solve(b);
    cout << "QR solution\n" << xqr << endl;

    cout << "Difference\n" << xplu - xqr << endl;
    cout << "For partialy pivoted LU Ax-b=\n" << a*xplu-b << endl;
    cout << "For QR Ax-b=\n" << a*xqr-b << endl;

    return 0;
}
```

EIGEN – РАЗЛИЧНЫЕ МЕТОДЫ РЕШЕНИЯ

```
Matrix A
 8.1041  0.0083 -3.1433
 1.7114   -0.02 -0.6638
   4.28 -0.0078   -1.66
Partial pivoting solution
-58282.2
-265.251
 -150263
Full pivoting solution
-58282.2
-265.251
 -150263
QR solution
-58307.2
-265.329
 -150328
Difference
  25.0078
0.0785522
  64.4844
For partialy pivoted LU Ax-b=
         0
-0.0078125
 -0.015625
For QR Ax-b=
  0.03125
        0
-0.015625
Press <RETURN> to close this window...
```

EIGEN – РАЗЛИЧНЫЕ МЕТОДЫ РЕШЕНИЯ

ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СЛАУ

- Вместо точного решения мы ищем приближённое
- Мы преобразуем исходную задачу к специальному виду:

$$Ax = b \implies \tilde{A}x = Bx + b \implies x = \tilde{A}^{-1}Bx + \tilde{A}^{-1}b \implies x = \tilde{B}x + \tilde{b}$$

где

$$A = \tilde{A} + B, \tilde{A} = diag(A)$$

- Тогда, если мы зададим начальное приближение x_0, получим итерационный процесс

$$x_{n+1} = \tilde{A}^{-1}Bx_n + \tilde{A}^{-1}\tilde{b}$$

- Метод простой итерации

СХОДИМОСТЬ ИТЕРАЦИОННЫХ МЕТОДОВ

- Всегда ли итерационные методы сходятся?
  - Сходимость основана на свойстве сжимающих отображений
    $$\rho(Ax, Ay) < \alpha\rho(x, y), 0 \leq \alpha < 1$$
  - Если отображение сжимающее, то метод сходится
  - Для каждого итерационного метода этот критерий свой
  - Например, для метода простой итерации $\left\|\tilde{B}x\right\| \leq \|x\|$
- Но, вот беда, мы не знаем x, как тогда проверить выполнение этого условия?
  - Можно построить оценку для произвольного элемента x, если воспользоваться согласованной нормой матрицы
    $$\|Ax\| \leq \|A\| \cdot \|x\|$$
  - Но тогда встаёт вопрос – что же такое норма?

НОРМЫ ВЕКТОРОВ И МАТРИЦ

- Мы будем понимать под нормой вектора его длину
- Под нормой матрицы мы будем подразумевать операторную норму

$$\|A\| = \max_{\|x\|=1} \|Ax\|$$

- Получается, что норма матрицы зависит от способа вычисления нормы вектора
- Используются т.н. нормы Гёльдера для векторов:

$$\|x\|_{l^p} = \left(\sum_{i=1}^{n} x_i^p \right)^{\frac{1}{p}}$$

- Зачем?

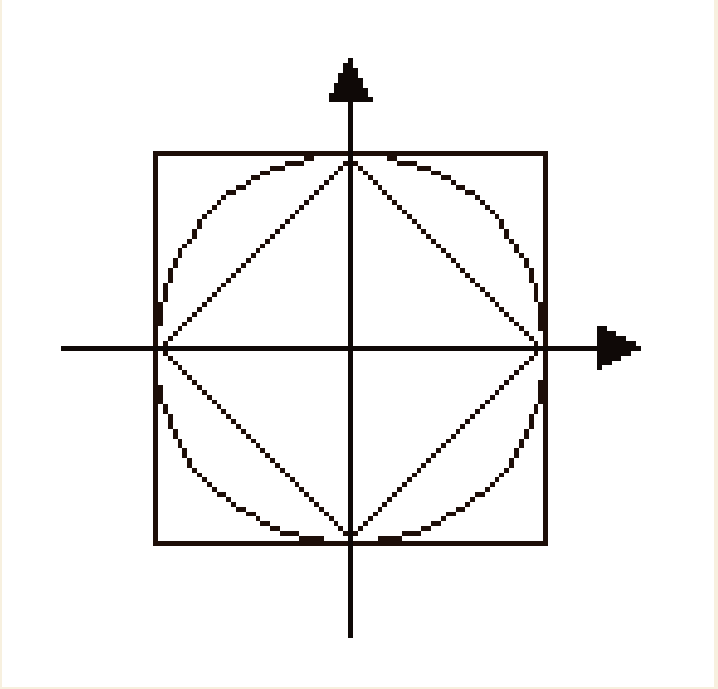
СОГЛАСОВАНЫЕ НОРМЫ

- Частные, наиболее распространённые случаи:

$$\|x\|_1 = \sum_i x_i \Longrightarrow \|A\|_1 = \max_j \sum_i |a_{ij}|$$

$$\|x\|_\infty = \max_i x_i \Longrightarrow \|A\|_\infty = \max_i \sum_j |a_{ij}|$$

$$\|x\|_2 = \sqrt{\sum_i x_i^2} \Longrightarrow \|A\|_2 =_1 (Ax, Ax) = \max \lambda_i$$



РЕАЛИЗАЦИЯ МЕТОДА ПРОСТОЙ ИТЕРАЦИИ

```
#include <iostream>
#include <eigen3/Eigen/Dense>

using std::cout;
using std::endl;

int main(int argc, char *argv[])
{
        Eigen::Matrix3d mat, A, B;
        mat << 6.25, -1, 0.5, \
                -1, 5, 2.12, \
                0.5, 2.12, 3.6;
        cout << mat << endl;
        Eigen::Vector3d b, b_orig, x,x_prev=Eigen::Vector3d::Zero();
        b << 7.5, -8.68, -0.24;
        b_orig = b;
        A = mat.diagonal().asDiagonal();
        B = -mat +A;
        for (int i=0; i<B.rows(); i++)
        {
                b(i)/=A(i,i);
                B.row(i) /= A(i,i);
        }
```

```
cout << "B=\n" << B << endl;
cout << "Norm of B=" << B.norm() << endl;
double normX=x.norm(), normXprev, tolerance = 0.0000001;
int counter = 0;

do
{
        x = B*x_prev+b;
        counter++;
        normXprev = normX;
        normX = x.norm();
        x_prev = x;
}
while(fabs(normX-normXprev)> tolerance);

cout << "After " << counter << " iterations we found X\n" << x << endl;
Eigen::Vector3d residual = mat*x-b_orig;
cout << "Residual norm " << std::fixed <<residual.norm() << endl;
Eigen::Vector3d xLU = mat.lu().solve(b_orig);
cout << "Same thing using LU-decomposition\n" << xLU<< endl;
return 0;
}
```

РЕАЛИЗАЦИЯ МЕТОДА ПРОСТОЙ ИТЕРАЦИИ

Для вычисления других норм используется функция

lpNorm<n>()

lpNorm<Eigen::Infinity>()

РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

```
Starting /Users/amakashov/projects/build-linsolve-Desktop-Debug/linsolve...
6.25   -1  0.5
  -1    5 2.12
 0.5 2.12  3.6
B=
        0      0.16     -0.08
      0.2         0    -0.424
-0.138889 -0.588889         0
Norm of B=0.786038
After 29 iterations we found X
0.8
 -2
  1
Residual norm 0.000000
Same thing using LU-decomposition
 0.800000
-2.000000
 1.000000
/Users/amakashov/projects/build-linsolve-Desktop-Debug/linsolve exited with code 0
```

ДОСТОИНСТВА И НЕДОСТАТКИ

+ При большом размере матрицы позволяют находить решение быстрее прямых
+ В них меньше накапливается вычислительная погрешность
+ Позволяют «управлять» точностью получаемого решения

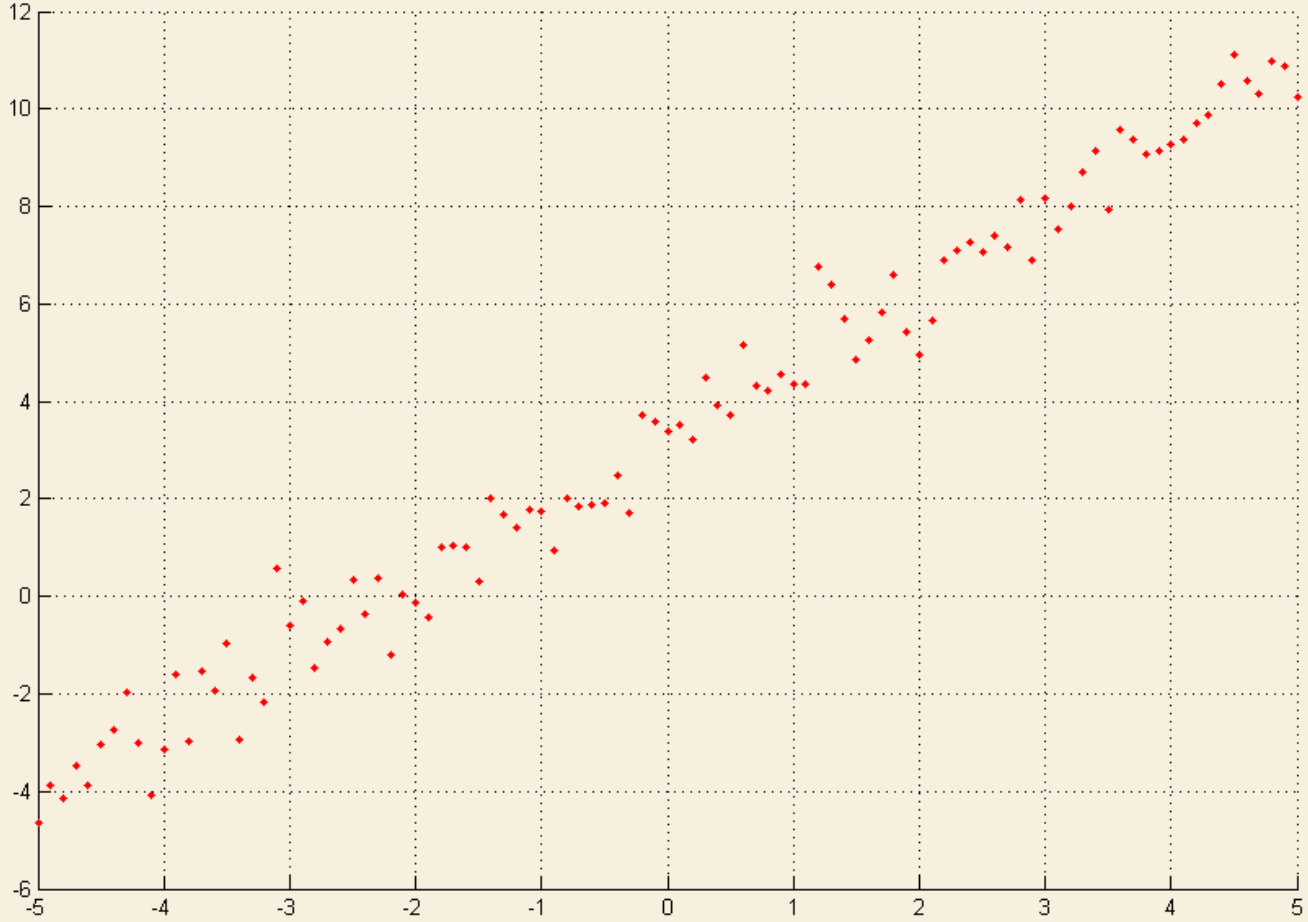
- Не всегда применимы
- Для матриц небольшого размера – нерационально использовать
- В некоторых случаях – требуется найти «хорошее» приближение

- Eigen::Matrix3d low,up;
- low = B.triangularView<Eigen::StrictlyLower>();
- up = B.triangularView<Eigen::Upper>();
- cout << up << endl << endl << low << endl;
-

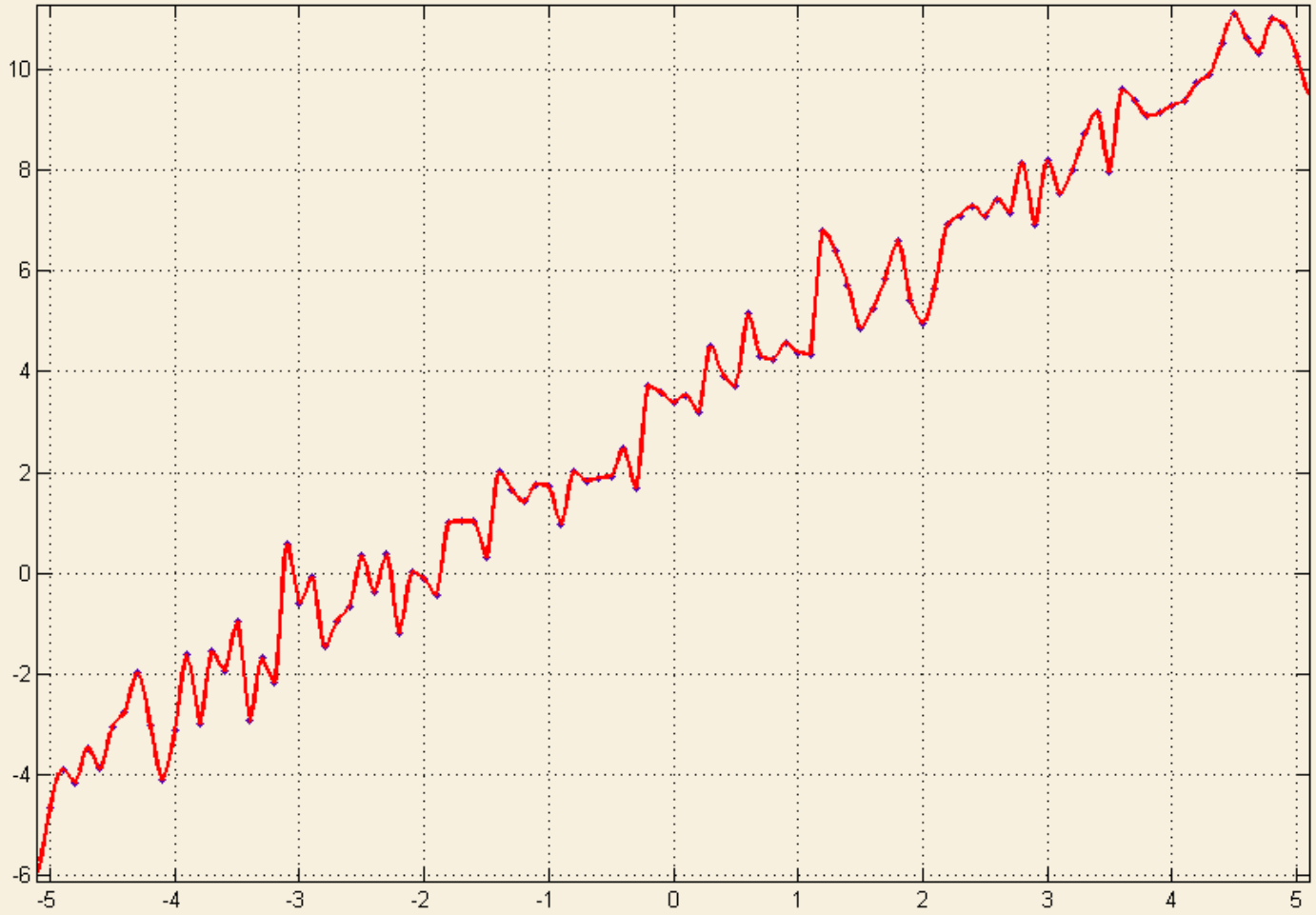
Метод Зейделя

$$x_{n+1} = \tilde{A}^{-1}B_l x_{n+1} + \tilde{A}^{-1}B_u x_n + \tilde{A}^{-1}\tilde{b}$$

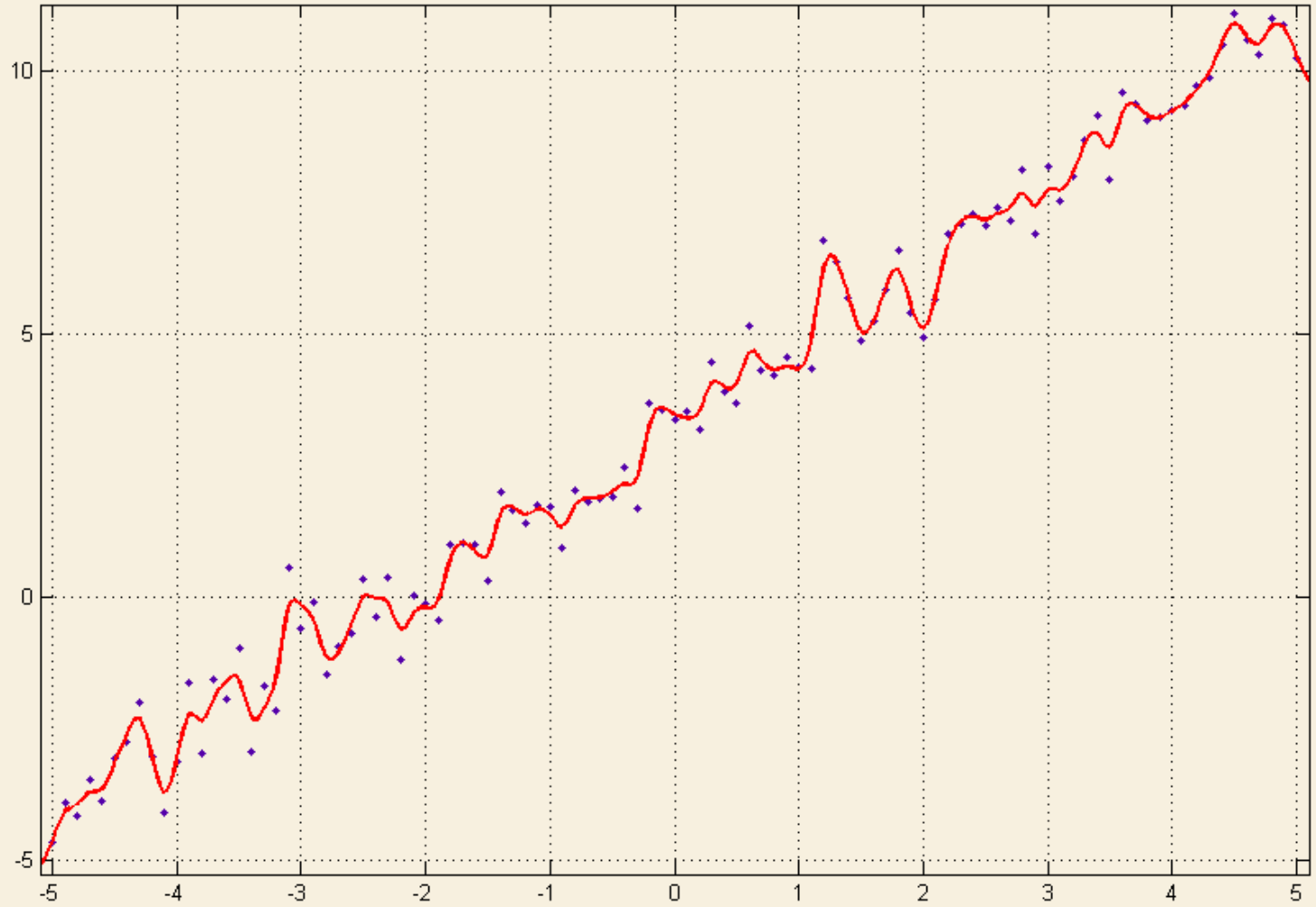
ОБРАБОТКА «ЗАШУМЛЁННЫХ ДАННЫХ»



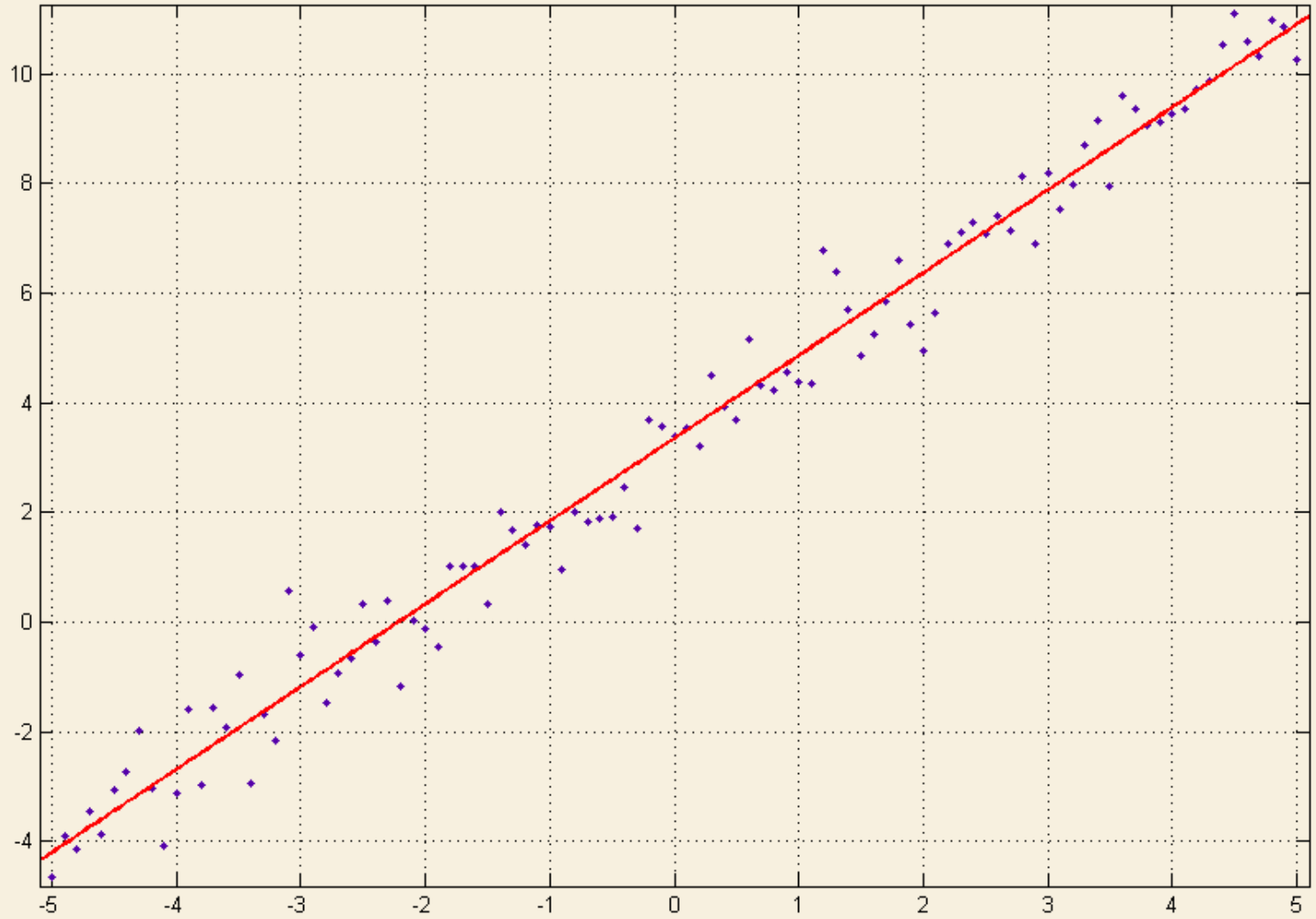
ОБРАБОТКА «ЗАШУМЛЁННЫХ ДАННЫХ»



ОБРАБОТКА «ЗАШУМЛЁННЫХ ДАННЫХ»



ОБРАБОТКА «ЗАШУМЛЁННЫХ ДАННЫХ»



ОБРАБОТКА «ЗАШУМЛЁННЫХ ДАННЫХ»

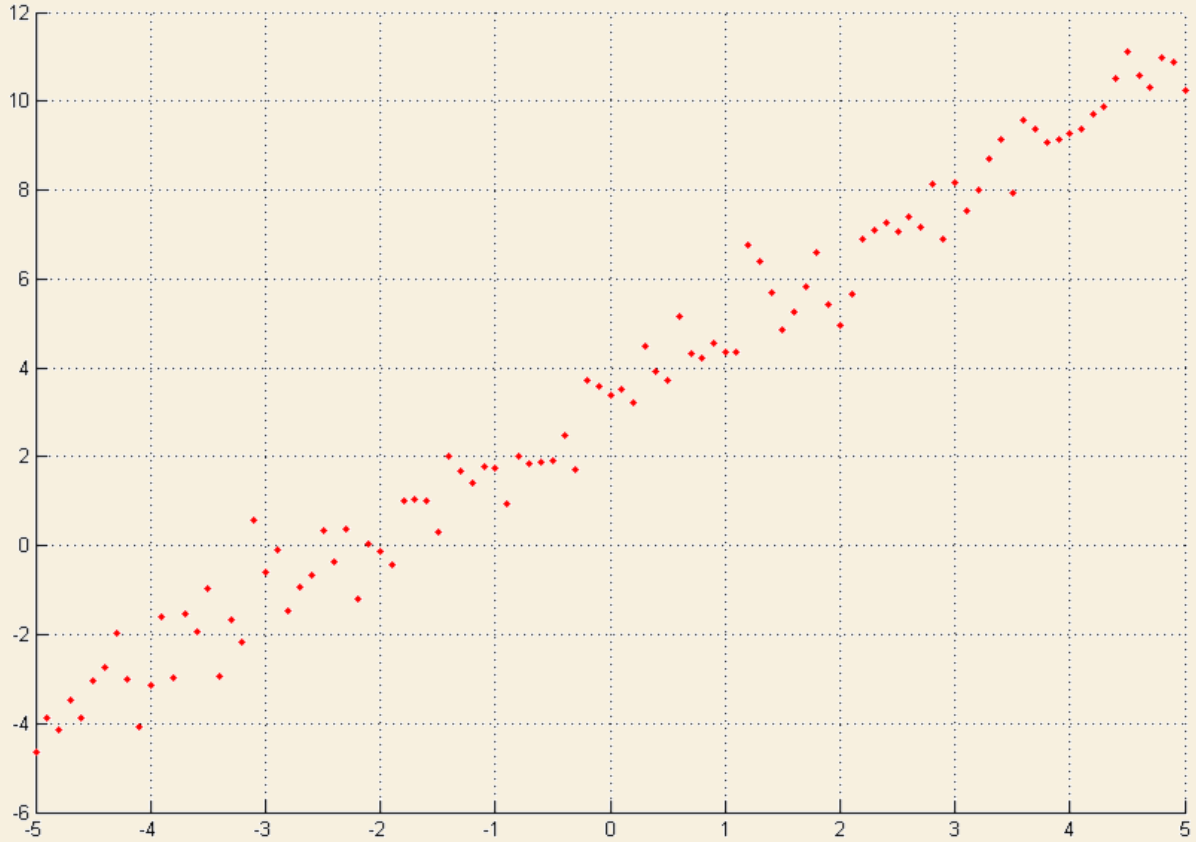
- Как выбрать правильное приближение?
- Как оценить его параметры?
- Как понять, что получилось?

МНК-ОЦЕНКА

- Используется для линейных по параметрам моделей
- Подразумевает, что модель мы каким-то образом уже выбрали
- Например:

  x – данные

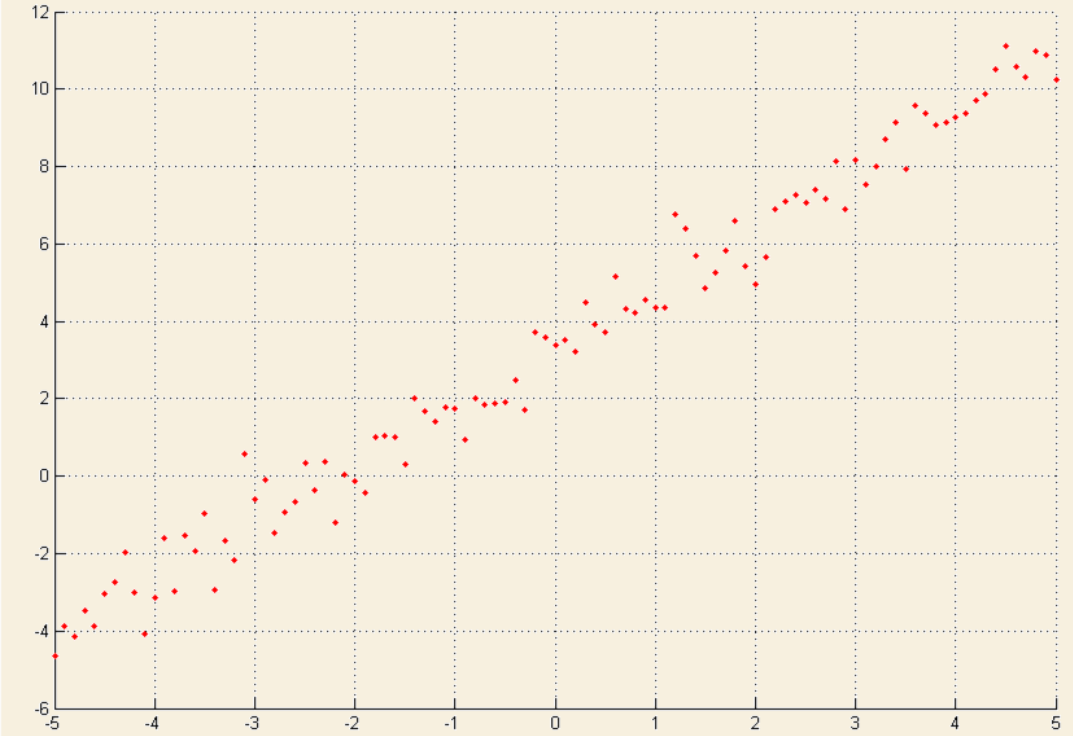
  y - значения



МНК-ОЦЕНКА

- Модели могут быть разными:

$$u = C = const$$
$$u = A \cdot x + B$$
$$u = A \cdot x^2 + B \cdot x + C$$
$$u = A \cdot e^{Bx}$$
$$u = C_1 \cos(\omega_1 x)$$
$$+C_1 \sin(\omega_1 x)$$



МНК-ОЦЕНКА

- Выберем линейный полином:

$$u = A \cdot x + B$$

- Получим набор значений $u_i = A \cdot x_i + B$
- При этом у нас есть y_i - практически полученные результаты
- Очевидно, что в идеальном случае $u_i = y_i$
- В реальности $r_i = u_i - y_i$ – невязка
- Нужно выбрать A и B так, чтобы невязка была минимальной

МНК-ОЦЕНКА

- Будем минимизировать квадратичный функционал:

$$\sum_i r_i^2 \to min$$

- Можно записать через вектор:

$$r^T r \to min$$

- Условие минимума – равенство нулю первой вариации:

$$\delta r^T r = 0$$

МНК-ОЦЕНКА

- Запишем уравнения для компонент r_i:

$$r_i = x_i \cdot A + 1 \cdot B - y_i$$

- Можно записать матричное уравнение:

$$\begin{pmatrix} \cdots \\ r_i \\ \cdots \end{pmatrix} = \begin{pmatrix} \cdots & \cdots & \cdots \\ x_i & 1 & y_i \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix}$$

- Тогда функционал можно записать как

$$\begin{pmatrix} A & B & -1 \end{pmatrix} \begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \\ \cdots & y_i & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots & \cdots \\ x_i & 1 & y_i \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} \to min$$

- После этого первую вариацию можно записать как

$$\begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \\ \cdots & y_i & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots & \cdots \\ x_i & 1 & y_i \\ \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

- То же самое

$$\mathrm{F} \cdot \begin{pmatrix} A \\ B \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

где $\mathrm{F} = \begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \\ \cdots & y_i & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots & \cdots \\ x_i & 1 & y_i \\ \cdots & \cdots & \cdots \end{pmatrix}$-матрица 3x3

МНК-ОЦЕНКА

- У нас 3 уравнения, но всего 2 переменных
- Можно преобразовать у уравнению 2x2:

$$\begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \end{pmatrix} \begin{pmatrix} \cdots & \cdots \\ x_i & 1 \\ \cdots & \cdots \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} \cdots & x_i & \cdots \\ \cdots & 1 & \cdots \end{pmatrix} \begin{pmatrix} \cdots \\ y_i \\ \cdots \end{pmatrix}$$

- То же самое:

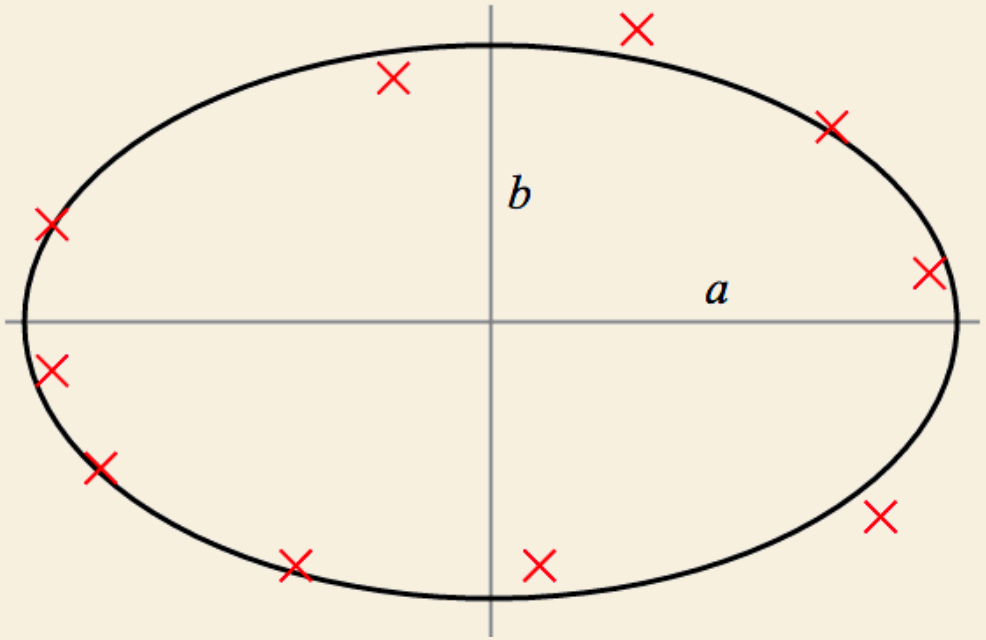
$$G^T G \begin{pmatrix} A \\ B \end{pmatrix} = G^T y$$

ЛИНЕЙНОСТЬ МОДЕЛИ

- Уравнение эллипса в общем виде

$$Ax^2 + By^2 + Cxy + Dx + Ey + 1 = 0$$

- Кажется, что уравнение квадратное, но по параметрам оно линейное
- Такое уравнение справедливо для каждой(x_i, y_i)



МНК-ОЦЕНКА

- А если модель экспоненциальная?

$$u = A \cdot e^{Bx}$$

- Можно получить линейную логарифмированием:

$$\ln u = \ln A + Bx$$

- Существует обобщение метода на нелинейный случай (NLLS)
- Тогда строится итерационный процесс с начальным приближением

$$\delta r^T r = 0$$

ПРИМЕНЕНИЕ В EIGEN

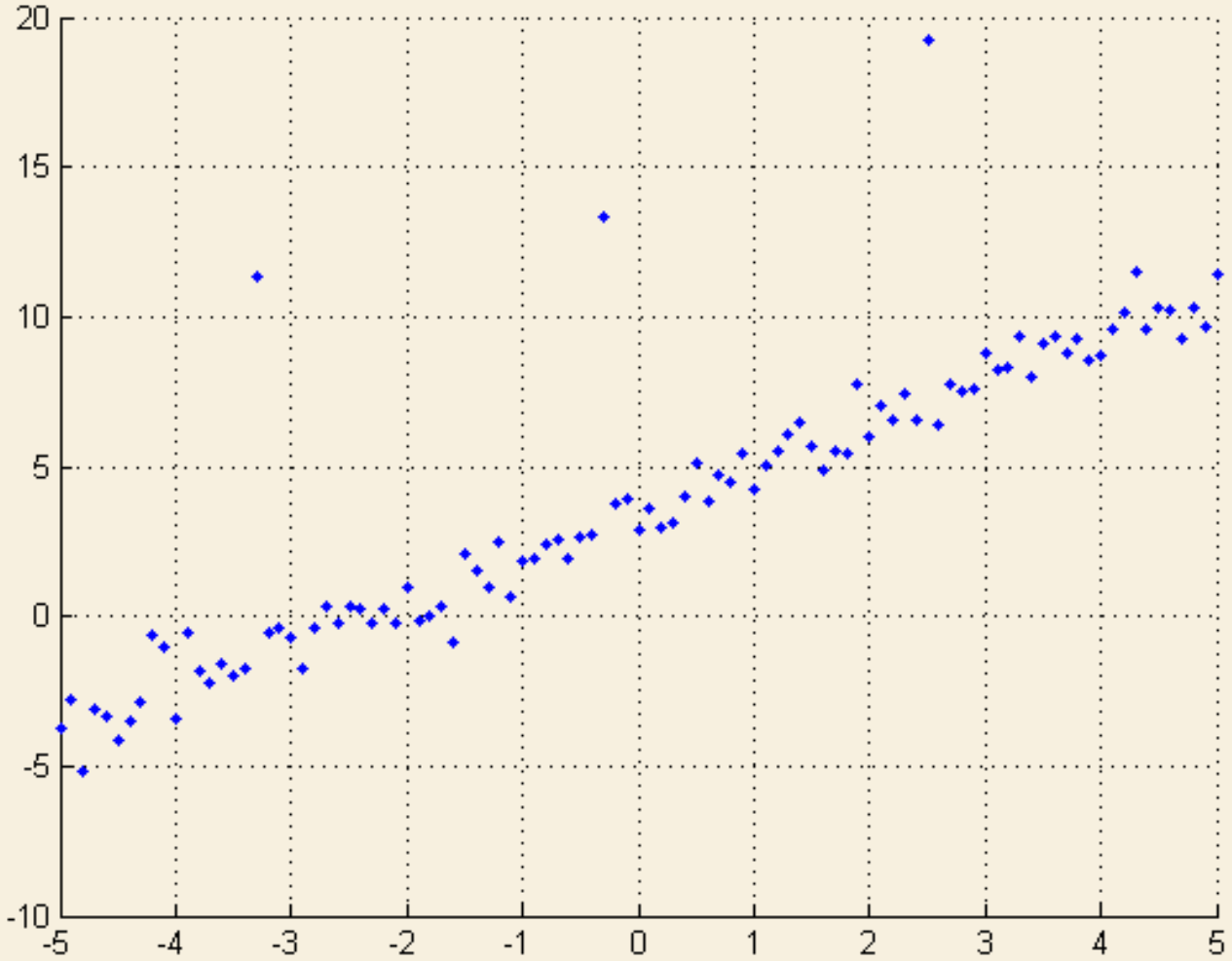
```
// Создаём матрицу и вектор со случайными значениями
Eigen::MatrixXf A = Eigen::MatrixXf::Random(3, 2);
Eigen::VectorXf b = Eigen::VectorXf::Random(3);

// Применение "в лоб"
cout << "The solution using normal equations is:\n"
<< (A.transpose() * A).lu().solve(A.transpose() * b) << endl;
// Более правильное (через QR-разложение)
cout << "The solution using the QR decomposition is:\n"
<< A.colPivHouseholderQr().solve(b) << endl;
```

МНК-ОЦЕНКА

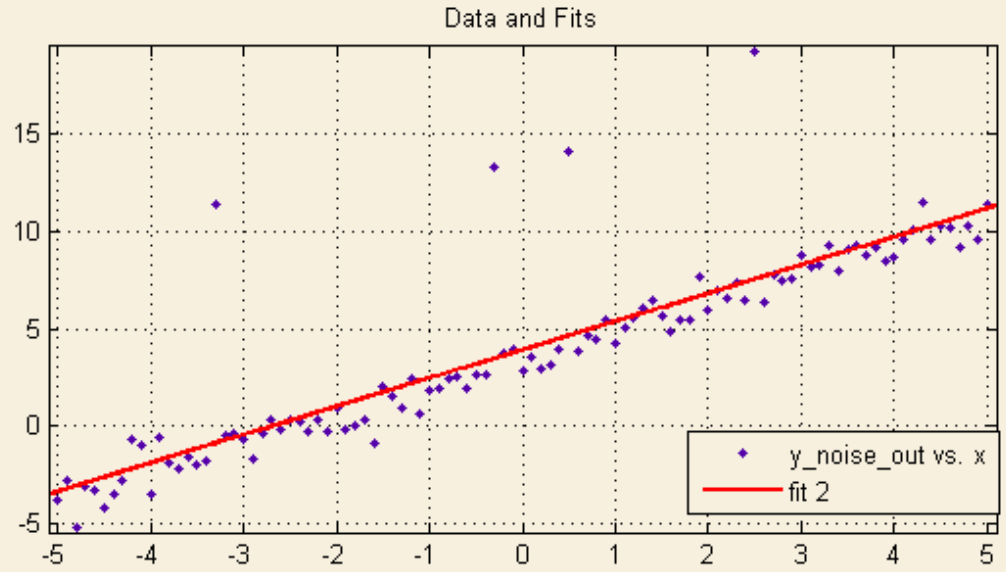
- Как оценить качество полученной оценки?
- Самый простой вариант – посчитать норму r_i
- Однако её величина зависит от размерности вектора (числа *n*)
- Выход – использовать отношение нормы невязки к норме правой части:

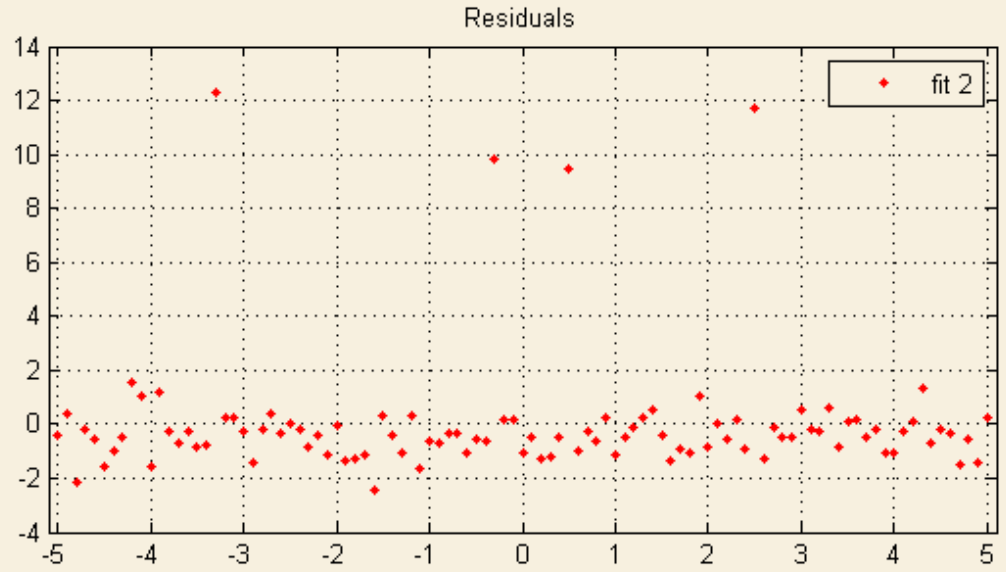
  $\delta r = \frac{\|r\|_p}{\|b\|_p}$
- Ещё одна проблема – оценка данных с выбросами (**outliers**)



ДАННЫЕ С ВЫБРОСАМИ

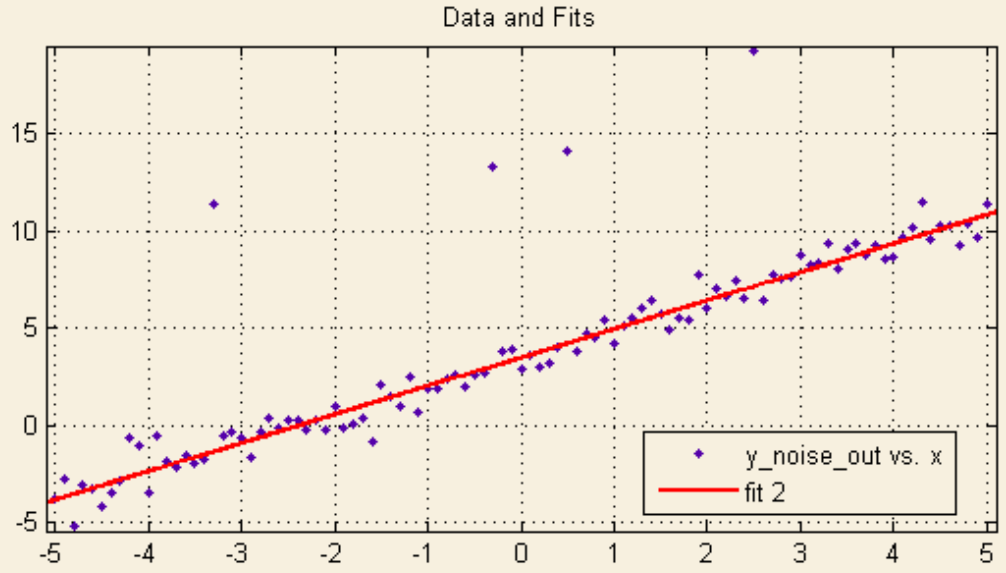
Та же самая линейная зависимость, но теперь в ней есть несколько «неправильных» точек

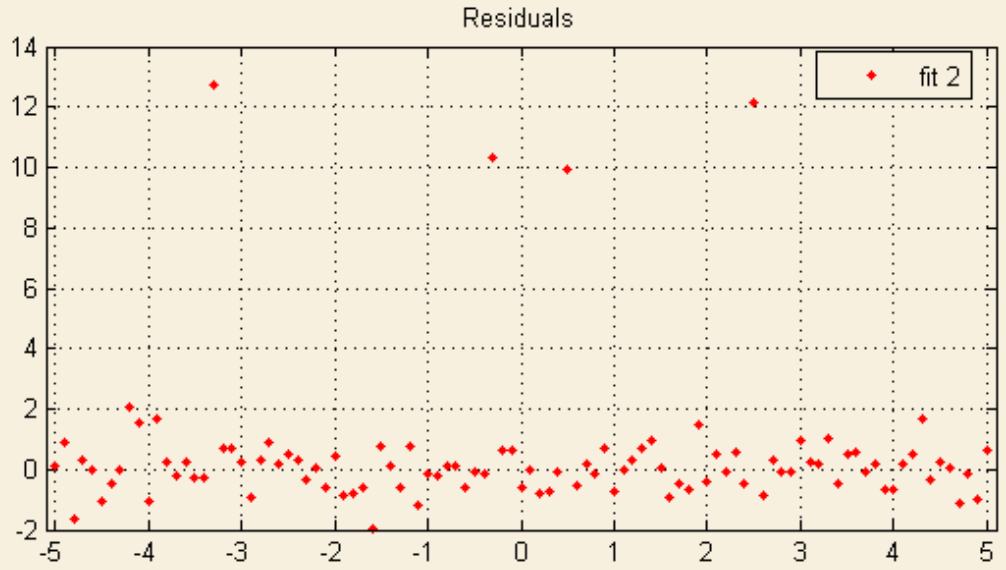




ДАННЫЕ С ВЫБРОСАМИ

Без учёта выбросов





ДАННЫЕ С ВЫБРОСАМИ

С учётом выбросов

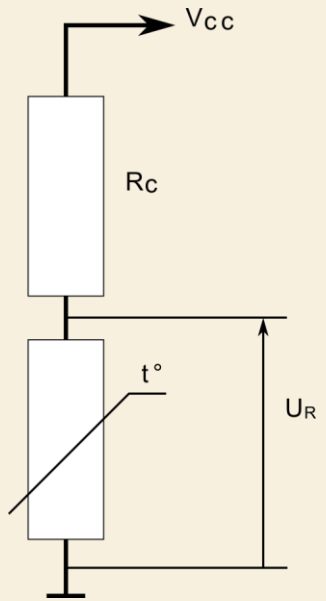
ДАННЫЕ С ВЫБРОСАМИ

- Необходимо использовать робастные методики оценки для исключения выбросов
- Один из вариантов - RANSAC
- Идея в том, чтобы случайным образом выбирать элементы из выборки для построения оценки

RANSAC

- Например: мы оцениваем 2 параметра по 100 отсчётам
- Выберем 100 пар из двух случайных элементов (x_i, y_i)
- Для каждой пары построим δr_j - оценку для j-гипотезы
- Для лучшей пары выберем точки с отклонением больше предельного δ
- Для оставшихся точек построим МНК-оценку
- Вообще говоря, оценку j-гипотезы надо строить тоже по МНК, выбирай заданное число отсчётов

КАЛИБРОВКА AVR ДЛЯ ОТОБРАЖЕНИЯ ТЕМПЕРАТУРЫ ТЕРМОРЕЗИСТОРА



- Показания индикатора

$$Z = 1024\frac{U_R}{U_{\text{оп}}} = 1024 \cdot \frac{R}{R_c + R}$$

$$\approx 1024 \cdot \frac{R}{R_c + R_0}$$

- Зависимость R - экспоненциальная

$$R = R_0 \cdot 10^{K(T-T_0)}$$

КАЛИБРОВКА AVR ДЛЯ ОТОБРАЖЕНИЯ ТЕМПЕРАТУРЫ ТЕРМОРЕЗИСТОРА

№	Z	T,°C
1	27	71
2	31	64
3	43	52
4	58	41
5	69	33
6	86	23
7	102	17
8	111	12
9	122	2
10	137	0
11	18	87
12	176	-5